Applying Web-based Networking Protocols and Software Architectures for providing adaptivity, personalization, and remotization features to Industrial Human Machine Interface Applications

Alessandro Bozzon, Marco Brambilla, Piero Fraternali, Paolo Speroni, and Giovanni Toffetti
Politecnico di Milano, Dipartimento di Elettronica e Informazione, Italy
{bozzon, mbrambil, fraterna, paolo.speroni, toffetti}@elet.polimi.it

Abstract

This paper proposes an innovative use of a mix of networking standards and software implementation technologies for the design of industrial Human Machine Interface (HMI) systems. We describe how well known technologies and practices can be transferred from internet-based architectures to embedded systems. We analyze the technologies that can be fruitfully used in the implementation of HMI architectures and illustrate the design of a real industrial HMI system that exploit internet communication protocols and Web-based architectures. Several advanced features can be achieved thanks to this architecture, such as application adaptivity, interface personalization, control remotization, and multi-channel notification. Finally, we evaluate the resulting platform in terms of performance, reliability, and usability.

1. Introduction

The current status of the industrial HMI (Human Machine Interface) in the field of industrial automation is characterized by a predominance of embedded low-power devices, typically comprising LCD touch-screens and/or keyboards, that are interfaced with proprietary or standard field buses, specifically devised for the industrial plant monitoring and automation. Commercial systems typically rely on proprietary architectures for the hardware and the operating systems, the I/O interface, the communication protocols implementation, the graphic display management, and the business logics. This situation is largely due to the strong focus on costs, performances and reliability, which largely overcomes the interest in standard architectures and high quality of interfaces and services. Moreover, industrial automation communication protocols have not reached the same level of standardization as office communication networks, which further justifies the predominance of proprietary architectures. Even if we consider only the networking standards, the industrial field can feature hundreds of would-be standard protocols, without any clearly predominant solution.

However, the success of the Internet and of the Web has started impacting the industrial HMI world too. Industrial users are starting to familiarize with Web interfaces, graphical quality, multimedia content, and features such as mobility, adaptivity, and personalization of the applications. At the same time, TCP-IP based communication protocols and embedded operating systems have started to spread in the industrial automation field, thus reducing the need of proprietary architectures and makes enterprise-wide integration more appealing. In this scenario, it is easy to foresee a slow but inexorable convergence of the industrial HMI solutions towards standard architectures, standard communication protocols, and advanced interactive functions.

Our work focuses on the design of a new distributed software architecture for HMI systems able to provide features and services such as personalization, adaptivity, distribution, mobility, multi-channel notification, integration with office networks and software packages, although preserving the robustness, reliability, performance and cost-effectiveness of traditional HMI solutions. The project, called ESA-MyHMI is a research activity carried out in collaboration between Politecnico di Milano and ESA Elettronica S.p.A., an Italian company operating in the HMI market. The project has lead to a novel HMI architecture, which leverages the most advanced architectural patterns of multi-tier Web applications to deploy sophisticated HMI functionalities on top of low-cost, industrial-class, embedded hardware.

The paper is organized as follows: Section 2 presents the current status of the HMI market; Section
3 states the requirements of the myHMI project; Section 4 discusses the major design decisions involved in the architecture specification; Section 5 focuses on the configuration of the HMI projects; Section 6 presents the implementation experience; Section 7 evaluates the results of the performance tests; Section 8 reviews the related work; and finally Section 9 draws the conclusions and highlights the future work.

2. Overview of the industrial HMI market

The HMI market is intrinsically a slowly moving world. Most of the market offer and request is still related low-end product categories, that are very well established in the market and feature only basic technical and functional solutions, sometimes implemented by obsolete technologies and approaches. Industrial HMI products rarely implement innovative services, such as remote access to the plant control, messaging and remote notification. Indeed, HMI companies seem to privilege exclusively performance and good access to industrial communication standards, even if these factors could be incompatible with the adoption of innovative solution based on modern and solid Web architectures. Even the HMI players that seem to offer the most innovative contents (and claim their products as Web-enabled) still leverage on legacy architectures, typically exploiting monolithic applications that provide poor flexibility and personalization features to system architects and users. Products tagged as “Web-enabled” in current HMI market that simply provide, as a matter of fact, applications natively available in the Windows CE or Windows XP Embedded operating systems (e.g., Internet Explorer, HTTPD web server, mail client, Messenger, and so on) without any real additional value.

On the contrary, recent studies [7] show how users are increasingly looking towards a new range of products with advanced features, superior graphical capabilities and improved usability that could grant:

- remote and, possibly, distributed control of an industrial plant, by providing all those characteristics that directly or indirectly enable a system to be remotely controlled through the use of network services (thin browser’s plug-ins, PDA/Smart etc. etc.);
- novel remote notification solutions for the occurrence of an event or of an alarm even when the user is not in front of the terminal;
- personalization and automatic adaptation of the GUI to allow users to customize the displayed information (e.g., alarms, screen data, widgets, and so on) and the graphic properties of the interface (e.g., colors);
- integration with existing enterprise processes, systems and equipments (e.g. IT infrastructure, fieldbus architecture etc.etc.);
- openness to new standard and best practises in the field, by offering low cost modularity and extensibility.

SCADA (Supervisory Control And Data Acquisition) systems recently introduced some interesting innovations but, as the acronym suggests, their target is focused on products that implement a wide range of high-level functionalities and that can be deployed in a large set of contexts. They are typically used for human-machine interfaces to be deployed on high-profile devices (PCs and powerful embedded systems) and represent a niche in the HMI market.

In the other market’s sectors, innovation has been lead by main vendors (e.g., Siemens), who have been working for the past few years in raising the level of the features provided by traditional HMI applications. Sm@rtAccess [20], for example, is a technology developed by Siemens that allows distributing the control of an industrial plant over a maximum of three stations. Its functioning, though, it is based on Sm@rt devices that simply broadcast the displayed image of the apparatus that is directly connected with the plant to the others clients. As a drawback, a similar approach bursts the amount of required transfer bandwidth, overcoming the capability of a typical Internet connection; moreover, Sm@rt technology simply exploits a Windows CE built-in application, available also on competitors’ products, which means that the technical added value is very low. Finally, its benefit can be exploited only for industrial plants having several Sm@rtAccess-ready HMI stations. Progea [21] proposes a more innovative solution by offering remotization features and a Web-based architecture. Running the Progea server application on a Windows XP based PC, it is possible to remotely control a plant from an internet connected standard Web browser that have the support of a JVM (Java Virtual Machine). Even if powerful, this solution lacks in offering a portable solution since, as Progea reports in its website, “because Java engine is not so reliable on WinCE, Progea has developed a special Client component, to ensure access to the Server also from WinCE stations”, thus offering different implementations for different platforms.
3. Requirements for novel HMI solutions

The market of industrial HMI is seeing a slow but steady evolution towards the integration of industrial automation terminals with software and hardware architectures typical of office and Web-based applications, to achieve greater usability and flexibility of the interface and easier interoperability between industrial automation solutions and enterprise information systems. This goal requires unbundling the functions and modules of a traditional HMI solution, deploying them over a modular and distributed system, which exploits the open standards of the Internet and the architectural patterns of multi-tier Web applications.

The myHMI project aims at designing, implementing and evaluating a distributed HMI platform enabling the definition of remotizable, personalized, and adaptable multi-device human-machine interfaces, which can be seamlessly accessed both locally and remotely and can be easily integrated in the enterprise ICT infrastructure.

Table 1. Functional Requirements.

<table>
<thead>
<tr>
<th>Functional Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic configuration</td>
</tr>
<tr>
<td>The organization and appearance of the HMI should not be hard-wired, but dynamically configurable in terms of number and type of the controlled variables, layout of the pages, displayed data, and so forth.</td>
</tr>
<tr>
<td>User login and access control</td>
</tr>
<tr>
<td>Users should be identified univocally, and granted access to the system based on a successful authentication. Access control should include page access and single object interaction control.</td>
</tr>
<tr>
<td>Personalization</td>
</tr>
<tr>
<td>The user should be able to customize (pieces of) the displayed information and the graphic properties of the interface, and save his preferences in a profile.</td>
</tr>
<tr>
<td>Interface adaptation</td>
</tr>
<tr>
<td>The user interface should adapt itself to fit the screen of heterogeneous devices by means of declarative rules.</td>
</tr>
<tr>
<td>Alarms management policies</td>
</tr>
<tr>
<td>The system should provide mechanisms for the notification of the alarms to the user, according to specific policies.</td>
</tr>
<tr>
<td>Functional restriction</td>
</tr>
<tr>
<td>The producer of the HMI system should be able to disable selected functions on specific terminals, for tuning the features on the product commercial value.</td>
</tr>
<tr>
<td>Reporting</td>
</tr>
<tr>
<td>Reports from log data should be produced in different formats, to allow remote visualization, dispatching and printing.</td>
</tr>
</tbody>
</table>

4. The MyHMI architecture

In this section we overview the main characteristics of the design of the MyHMI framework. The overall architecture of the myHMI platform is illustrated in Figure 1: the HMI functionality, usually embedded within the terminal attached to the controlled system, becomes partitioned into a client-server architecture, implemented on top of a hybrid communication network, comprising an Ethernet backbone that connects the HMI devices and a set of field bus protocols for connecting to the controlled plant.

Table 2. Non-Functional Requirements.

<table>
<thead>
<tr>
<th>Non-Functional Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network topology</td>
</tr>
<tr>
<td>The system should support a network architecture for standalone, local (LAN), fixed remote (wired Web) and mobile remote (Wireless) access. Client-server communication should exploit the HTTP protocol, for firewall compatibility.</td>
</tr>
<tr>
<td>Software architecture</td>
</tr>
<tr>
<td>The software should be based on standard operating systems (i.e., both Linux and MS Windows). The client application should run in a standard Web browser and should automatically scale on different screen resolutions; the server application should exploit a standard dynamic Web architecture.</td>
</tr>
<tr>
<td>Presentation</td>
</tr>
<tr>
<td>The interface should exploit device-independent rendition technologies (i.e., XHTML, SVG, Flash).</td>
</tr>
<tr>
<td>Performance</td>
</tr>
<tr>
<td>Performance of page data refresh should be comparable to standalone HMI systems (10 data refreshes per second). Furthermore, idle time after login should not exceed 3 seconds and the delay after a page switch should be less than 1 second.</td>
</tr>
</tbody>
</table>
4.1. General design choices
The design of the system had to address several issues, according to the requirements. In this section we summarize the issues and the adopted solutions.

4.1.1. Distribution model of presentation and business logic. The architecture design has been addressed by applying the state of the art solutions for granting separation of concerns and modular implementation. The possible solutions could be summarized in the following approaches:

1. **Monolithic architecture**: in this solution the whole application resides on the same device and is composed of modules that cannot work separately. This is the solution adopted by the traditional HMI applications. Its main advantage is the low consumption of resources. In contrast, this solution can provide only a low level of personalization, remotization and device independence.

2. **Pure web architecture**: this solution offers the typical features of a web application. This means that client and server features are separated and several clients can access the same server. The main drawback of this solution is the low quality of the interface and of the user interaction at client side (typical of traditional Web applications). Moreover, most of the computation is delegated to the server, included the preparation of the GUI and the management of the user’s input.

3. **Rich web interface**: this solution consists of extending the client side of traditional web architectures, thus moving some of the computation from the server side to the client side, which can be classified as a smart client interface. The business layer is still located at server side and contains the control policies (e.g., the interface adaptation and personalization rules, the alarm management and data logging policies, etc), while the presentation layer is implemented at the client-side. It is responsible of building the interface for the human supervisor and of managing the user interaction. The client side can leverage powerful rendering and interface technologies, such as Macromedia FLASH, SVG, Laszlo, or others.

According to the requirements, we decided to adopt solution 3. The abovementioned modularization of the architecture allows great flexibility, but also poses severe challenges to the overall performance. Several re-design cycles have been performed to guarantee that the partition of responsibility between the client and the server fulfilled the performance requirements and provided the best interaction capabilities.

4.1.2. Personalization solutions. One of the most challenging requirements advocated for strong personalization capabilities of the platform. In order to achieve this, three possible solutions could be exploited:

1. **Separate applications**: the more rude solution is to deploy a separate application for each user or group of users. This approach is not viable in terms of performances and modularity of the design, since it requires several processes to be running at the same time.

2. **Totally dynamic configuration**: this represents the opposite extreme in the spectrum of the solutions. It provides a unique implementation, which is so general that all the possible personalization rules are applied at runtime, once the user is recognized. This is a very powerful and flexible approach, however it encompasses huge resource consumption, in terms of computation power. Unfortunately, embedded systems cannot afford this kind of approach.

3. **Personalization based on groups**: an intermediate solution consists in providing a unique application with full-fledged personalization based on user groups. This approach assumes that users can be classified in groups/roles, thus factoring out most of the personalization rules at the level of the groups instead of the level of the single user. The remaining detailed personalization rules can be applied on the single user, but we may assume that their number and complexity are very limited. In this way, the application could exploit a set of pre-determined version of the interface based on groups, and then apply a small number of user-specific personalization rules.

The last solution implements a good trade-off between the needs of fine-grained personalization and of reducing the complexity of the computation.

4.1.3. Configuration. The application can be configured by means of a set of descriptors, which are edited offline by means of a Configuration Editor and uploaded to the server. The deployment descriptor contains all the details of the HMI project, including the interface layout, the list of controlled variables, the data to be visualized on each screen, the user access rights, the alarm management policies, and so forth. The configuration management can be implemented in two ways:

1. **Configuration files**: the HMI platform exploits a generic binary code that is configured at runtime by reading a set of XML descriptors containing the configuration rules for the specific application. This brings the advantage of having a unique binary code and easily readable and modifiable configuration files. On the other hand, this solution requires some additional computation and delays.
due to the file access and the application of the rules.

2. **Configured binary code**: this solution assumes to compile running code already configured for each specific application. This means that the offline Configurator Tool is in charge of building specific executable code every time the configuration process occurs. This allows to produce a much more efficient binary code for the HMI platform.

   The adopted solution comprises a mix of the previous options: the final architecture provides a server side application implemented according to solution 1, which means that the server side application exploits a unique binary code plus a set of XML configuration files. The client side part of the architecture is compiled in a unique running code, comprising several modules that embed also the configuration rules (page structure, adaptivity, personalization). The generated code is devised to be executed within browsers on any platform (embedded system, MS Windows PCs, Linux systems, and so on).

   This allows very efficient clients (whose code can be easily compiled offline thanks to existing technologies like Laszlo, that can compile client-side code starting from declarative specifications) and good server side performances (the access to XML files by server side programs is reasonably fast, and avoids the burden of building procedural code and recompiling it at configuration time).

4.1.4. **Connectivity.** The communication between client and server could be implemented by means of different protocols and techniques:

1. **TCP-IP bidirectional communication**: based on the management of sockets and allowing any kind of communication between the parties, without exploiting any web-specific behavior;
2. **HTTP interactions**: based on standard HTTP protocol and implementing client update by means of polling-like requests by the client to the server;
3. **HTTP with (emulated) callback**: implementing a simulated callback by the server to the client on data updates and other events, alarms, and so on.

   The adopted solution is the emulated callback implemented through the HTTP request-response cycle. The client submits requests upon user’s interaction and upon timeouts generated by its internal clock. Requests submitted to the server are left pending until an actual update on the status of the controlled system happens. In this case, the server sends a response to the client, thus simulating an event-based message sending. Upon reception of the request, the client submits another request. Notice that the server response contains only the field data and the updated configuration parameters, whereas all the other aspects of the page remain cached at the client and need not be exchanged.

   **4.1.5. Personalization enactment.** Some specific decisions must be taken about how and where to apply the personalization and adaptation rules on the interfaces.

   Personalization and adaptivity rules could be stored and managed with two approaches:

1. **Rules encoded as XML files**: personalization rules are generated by the offline configurator tool in XML format. Such rules are parsed and interpreted at runtime by a general purpose code, that generates the expected interface. A variant of the approach could devise several specific components for parsing the personalization and adaptivity rules addressing different issues (e.g., user interface, alarm configuration, and so on);
2. **Rules embedded in the source code**: this solution represents a hybrid approach, in which the rules are entirely applied at the server side and the client receives the information and the application structure description ready for the specific context (user and device);

   The options about where to apply the rules are:

1. **Server side only rule calculations**: the rules are entirely applied at the server side and the client receives the information and integrates it according to the context and the rules (that are sent as well);
2. **Client side only rule calculations**: this solution represents a hybrid approach, in which part of the rules are applied at server side and the remaining ones at client side.

   In our implementation, we decided for the third solution. We apply at client side the rules that affect the user interface and, in general, the client side issues. For this part, the rules have been stored as binary code in the client application, for performance reasons. Vice versa, we adopted server-side application of the rules concerning the server configuration. In this case, the rules have been encoded as XML files and parsed by the server components. This decision was mandatory since it was not allowed to generate the binary code for the rules to be applied at server side, because this would have needed to embed a C++ compiler in the configurator tool.
4.1.6. Access to field variables. The controlled system status can be made visible to the HMI system at different layers:

1. Directly to the client: the client checks the status directly on the field bus. This solution has the big disadvantage of losing the standard web configuration of the interactions;
2. Through a single centralized server: all the clients invoke a central server that acts as a gateway and provides the data about the plant status. This solution is acceptable for small systems whose status can be checked by one single entry point;
3. Through a server or a proxy: every client always invoke the same server, but several servers can be located on the plant to access different parts of the controlled system. The server invoked by the client may act as a proxy, by asking the needed information to the origin server that actually has access to the required field data.

We adopt solution 2 for simple single-server architecture, while we move to solution 3 in case of complex configurations. In any case, we chose to avoid clients that could directly invoke the field or several servers at the same time.

4.2. Design of the server

In the proposed MyHMI architecture, the server assumes the role of broker between the HMI interfaces and other servers that communicate on TCP/IP networks, and field buses connecting heterogeneous devices, possibly communicating through proprietary protocols. The server manages the connection to the field (via an OPC server module [1] or similar interfaces) and buffers the field data (in a data server module) to be delivered to the clients on the TCP/IP connection. Clients can be deployed in two configurations: locally at the server’s node (thus offering an integrated terminal interface) or remotely on a separate terminal connected to the server by means of a TCP/IP network.

The server manages three types of client requests: initialization requests, new page requests, data refresh requests, and event-triggered executions. Initialization and new page requests may require the computation of server-side personalization rules (typically those involving alarm management), which are processed by the server based on the identity of the requesting terminal and user; page data refresh requests involve only the shipping of raw data to the client and are served faster.

As depicted in Figure 2, despite such a variety of involved actors, the server identifies the boundary between two major classes of components. On one side there is the controlled system, composed by different devices, communicating both through industrial (e.g. Modbus, Fieldbus etc…) and web protocols (e.g. TCP/IP) and conveying data originating from the controlled environment; on the other side there are users, interacting with the controlled system with the support of a client user interface.

Acting as a broker, the server has to deal with a lot of challenging tasks like: (i) managing and coordinating the data flow between the involved actors, possibly performing ad-hoc data manipulation and aggregation; (ii) guaranteeing the synchronization of the status information at the different peers; and (iii) offering a secure and reliable service by ensuring fail-safe execution of user commands.

The server’s internal organization has been conceived to enhance modularity, extensibility, component re-use, and performance. In Figure 3 and Figure 4, we can identify three macro components of the server internal architecture:

1. the Field Interface Management (FIM);
2. the Control Interface Management (CIM);
3. the User Interface Management (UIM).

The FIM comprises all the server sub-components responsible for managing the communication with the field devices, while providing the abstraction and modularity required from other components to ignore the physical features, topologies, and protocols of the devices. Among these components, the most important is the Data Server: its responsibility is to control the system components used for all the input and output operations that have to be performed with field devices.
Interaction with the field is accomplished through a standard OPC client/server module, thus adding another level of abstraction (and modularity) to the system.

The CIM handles all the features related to user-command management, content personalization, and adaptivity. Since multiple users are allowed to interact with the system and user interface’s contents are directly related to user devices and authorizations, there is a need for ad-hoc data structures and operations able to comply with the performance, scalability, concurrency and reliability requirements. In order to respond to such demands, the internal organization of the CIM relies on orthogonal modules (included in the brokering control macro component) responsible for managing the communication with the FIM, each one dedicated to a single aspect.

The synchronization module takes care of granting correct concurrent execution of operations performed by different clients over shared objects (like field variables or alarm states) avoiding inconsistencies and, where needed, overcoming communication problems eventually leading to unstable system states. Working in collaboration with the synchronization module, the system actions component (Figure 4) provides an interface for the execution of commands over the system, by offering the whole set of operations made available by the system architect, like setting the value of a field variable, or acknowledging an active alarm.

The adaptation module, instead, is responsible for applying the personalization rules to the data to be dispatched to clients. Complementary to the adaptation module, the communication buffers component (Figure 4) responds to the requirements of performance required by MyHMI to perform multiple update of the system state to clients.

Every time a client connects to the system, a dedicated communication buffer is assigned to it and is initially filled with all the values needed to build up the current system state view for the client.

When the system produces a new event (like a new value, an alarm, and so on), the event instance is processed by the adaptation module and, if pertinent to the user personalization rules, is queued inside the buffers. The buffer, thanks to a publisher/subscriber registration pattern, notifies the upper levels of the architecture about the presence of new values. If possible, the upper level components update their state by retrieving (and flushing) their buffer. Thus, a punctual filtering of all the information directed to clients is applied at the source, thus reducing the amount of data managed by the system and, hence, improving performances. Client adaptation is also achieved by dimensioning the buffer size (and the buffer frame) dependently on the client device performance (this information is provided by clients during their connection). Moreover, to grant communication reliability, a buffer provides a verification mechanism over sent values by implementing a simple yet effective CRC control which checks, for every communication cycle (i.e., when clients empty their buffer) if the previous communication went right. In case of failure, the server sends back again the same information.

In addition, the CIM implements several advanced functions like multiple logging subsystems, possibly located on other network nodes (for example, dedicated PCs equipped with database systems), and multiple notification subsystems, allowing operators to receive notifications via short text messages on cellular phones, e-mail, or even instant messaging platforms. Again, the CIM is independent from both the field components and the user components, allowing re-using the same approach over different architectures both distributed (like Web Services or RPC call) and monolithic.

Finally, the UIM is the component delegated to orchestrate and synchronize the interaction with clients; since MyHMI relies on a Web architecture, the UIM is organized according to the Model View Controller (MVC) design pattern [5], in its Web-
compatible version known as Model 2 (MVC2) [6]. The MVC2 pattern separates the three essential functions of an interactive application: the business logic (the model, in our case the CIM), the interface shipped to the client application (the view) and the control of the interaction triggered by the user (the controller). The emitters of requests are clients. When a user executes an action on the user interface (like pushing a button or acknowledging an alarm), an HTTP request is sent to the server and caught by the UIM (controller), which in turn decides the course of action necessary to serve each request between: (i) executing an operation over the system or (ii) retrieving the updated system state. After completion, the CIM communicates the outcome of its execution to the UIM, which decides whether to execute other actions or to invoke a View component, responsible for formatting the results to send back to the client.

4.3. Client-server interaction design

An important aspect to consider when dealing with Web architectures is the asymmetric nature of the communication protocol used by clients to communicate with the server (HTTP). Only clients are able to perform requests to the server and not vice versa. This constraint hampers the optimization of the interactions because clients cannot be notified by the server of new events (new variable values, instructions, alarms), but need to periodically invoke the server to retrieve the updated information. Modern Web applications start to discover the usefulness of bidirectional communication mechanism and leverage on push technologies to enable Web server proactivity. According to the decision described in Section 4.1.4, we adopted a simulated callback approach. Such approaches usually rely on HTTP/1.1 persistent connection (see [22], [24], [25]). Persistent connections can now exploit the XMLHttpRequest concept [23] (originally developed by Microsoft as part of Outlook Web Access, a W3C working draft) and similar mechanisms to retrieve information from the server without necessarily updating the whole page. Thanks to these technologies, clients are allowed to establish a single connection always available for servers to send data independently from the user interaction. Unfortunately, persistent connections are expensive to manage for servers, making such approach unsuitable for low-computational-power devices like the ones used by MyHMI.

The communication buffer mechanism implemented in the CIM helps at overcoming the drawbacks coming from the polling process.

4.4. Client-side design

The main role of the client layer in the MyHMI architecture is to manage the data presentation and user interaction. Since we want our architecture to be independent from specific technologies, we produced a high-level design, which can be implemented in different rendering environments. The designed internal component distribution is depicted in Figure 6: the client application incorporates an application shell, executed within the browser environment. The shell (written in a client-side scripting language) is separate from the graphic engine of the browser and exploits a Model-View organization.

The Model contains the business objects of the interface (e.g., data variables, trend monitors) while the View comprises the widgets and presentation properties managed by the rendition technology (e.g., the widget to be used to display a data variable, a trend, or an input control).

The shell is responsible for managing the client application initialization by composing the user...
interface according with the information received from the server at the login or at the request of a GUI page; the initialization procedure builds a list of the model objects and page widgets and registers them as listeners to their relevant data variables, according to the building rules for the page (that include personalization and adaptation). According to the discussion of Section 4.1.5, client side rules are stored in a binary format.

The shell also manages an internal clock, used to automatically trigger requests to the server for the system state refresh; upon reception of server responses, the shell updates the internal data variables, which automatically refresh the registered business objects and associated widgets. This data-centric approach allows to redraw only the affected widgets, minimizing the computational effort and enhancing performances.

The shell also manages an internal clock, used to automatically trigger requests to the server for the system state refresh; upon reception of server responses, the shell updates the internal data variables, which automatically refresh the registered business objects and associated widgets. This data-centric approach allows to redraw only the affected widgets, minimizing the computational effort and enhancing performances.

5. System Configuration

We devised a highly configurable system where most of the components are configured during the setup stage by means of deployment descriptors, produced by an offline visual Configuration Editing tool. This section aims at highlighting the configuration mechanisms and solutions.

5.1. Server Configuration

Since the server stands between the controlled environment and the users, its configuration has necessarily to deal with both the users and the plant.

On one hand, the server collects all the information related to: (i) the number and type of all the field variables; (ii) the list of alarms the system should raise when a field variable assumes particular values; (iii) the topology of the controlled system, by having some references to distributed devices (like physical addresses, IP addresses...); (iv) the classes of client devices allowed to communicate with the server, as well as information related to their physical characteristics. On the other hand, the server also needs to be aware of information related to the organization of user interfaces, user authentication, user personalization, and so on.

The deployment descriptors containing such information are encoded as XML files [2], and are processed by the server using an event-based parser (more precisely, a SAX parser) to avoid any unnecessary object creation overhead during the server initialization step.

Even if this approach resulted to be suitable in terms of performance, the presence of a large number of configuration files (about one hundred files for a typical middle-size project) suggested us to find an
alternative way to retrieve information while keeping XML as reference file format. We found suitable an approach based on plain text index files associated with each XML file: such indexes contain absolute references to XML tags inside the XML documents, allowing fast sequential readings, thus improving performances.

The most complex configuration aspects are involved in the content personalization process: the CIM’s adaptation module described in Section 4.2 needs to be fed by information regarding the pages currently displayed by the client, about visual objects related to field variables (e.g. trend graphs, gauges or simple text fields), to alarms and, in general, to the entire set of variable managed by the server. However, due to personalization and adaptivity rules, the properties of the objects placed inside a page can change, depending on the devices (screen size and computation power) and on different authentication levels of the users. Of course, personalization and adaptivity rules could apply simultaneously, thus producing an actual number of rules equal to their Cartesian product. In order to keep minimal the total amount of information processed by the server, we decided to implement a two-step personalization algorithm. During the system design, the configuration editing tool produces:

1. an XML descriptor file containing, for each page, the information associated with the most “general” page configuration (i.e., when the page is displayed on the most powerful device from a user having the highest authentication level).
2. a “delta” xml file for each allowed client device that contains, for each user role, only information about the objects to modify or remove from the page.

When a user logs in the system, information about its device and its credential are collected; on a page request, personalization and adaptivity “delta” rules are applied on the general XML configuration file, thus producing the final rule set used by the CIM’s adaptation module.

Since XML parsing and transforming are know to be expensive tasks, we decided to adopt a more pragmatic approach by performing memory sequential reading of XML files as if they were plain text (thanks, again, to index files) and applying directly the personalization rules originating from the “delta” transformation files.

5.2. Client Configuration

While the server focus is on managing all the information related to the business logic of the system, the client configuration file should contain parameters and instructions about (i) how to populate every single page of the user interface and (ii) about how to communicate with the server (and what to communicate).

The configuration of the client leverages a client-side configuration file, which is downloaded at each user login and at each request for a new page. It contains all the relevant client-side personalization rules.

The information needed by the client for building the user interface includes: (i) the list of all the objects composing the user interface as well as their graphical properties; (ii) the identifier of the graphical skin used by the client instances; (iii) the page to display when a user logs in; (iv) for a logged user, the personalization rules to apply to each GUI page; (v) for a logged user, the default interface language.

The amount and variety of data needed by the client requires optimized approaches in order to avoid redundancies and to reduce the page loading time. The application of adaptation and personalization features is a resources-consuming task. The chosen approach heavily leverages on the Configuration Editing Tool which, at compile time, creates $n$ different version of the binary client application, each one tailored on a particular couple of user and device classes: when a user logs-in, its authentication information are communicated to the client application, which retrieves the proper pre-compiled user interface modules from the server. Information about the interaction between the client and the server (like the format of the request to perform) are also pre-compiled inside the downloaded modules. Language data are downloaded and applied on demand when (and if) the user requires a language change with respect to the default language of the project.

This approach has the drawback of introducing a lot of pre-compiled client modules, but dramatically reduces the amount of time needed at run-time to adapt the GUI to the user and the device, thus enhancing the reactivity of the system.

6. Implementation

This section reports our experience and evaluations in implementing a running prototype of the proposed MyHMI solution. Thanks to an open-source Modbus simulator, we simulated the logic and data flows of a milk bottling plant, composed of about twenty controlled variables, such as the liquid level and temperature of the milk tank, the state and speed of the automatic conveyor belt, and so on.
6.1. Server-side implementation

The first implementation choice was about the operating system to adopt and the possible technologies that could be associated to it. We evaluated both Linux-based and MS Windows-based operating systems, and finally we decided to use Microsoft Windows CE 4.2 [4] and its built-in technologies: Web publishing architecture rely on the ISAPI+HTTPD daemon, all the model and business logic components have been developed as Microsoft COM objects, and the controller part of our MVC architecture has been implemented as an ISAPI DLL.

The choice of leveraging on built-in Windows CE technologies has been mainly driven by performance considerations since alternatives approaches (as embedded Java-based Web servers) resulted to be excessively resource consuming. For the same reasons, we discarded advanced solutions like ASP, .NET or Web Services technologies. Some of them were not available on embedded devices, while others produced unsatisfactory results.

The connection with the field has been achieved thanks to a run-time component (called OPC client) based on the OPC Data Access Custom Interface Specification 3.0, coupled with an OPC Server version 2.5.15. The OPC client interfaces the Data Server component to the OPC server, which in turn manages the field connections. OPC Client/Server connection is open at the start-up of the system and it is kept alive until the server releases the resources before being switched-off. Thanks to the OPC architecture, we could raise the level of abstraction of the communication to the field, thus developing a prototype compatible with almost all the industrial field protocols. However, in future optimized implementations, we are planning to remove the OPC layers, in order to achieve better performances with low-power devices.

6.2. Client-side implementation

The implementation strategy for the client side has been heavily affected by the constraints imposed from the target environment of the MyHMI project. Accordingly to the choices made for the server side implementation, we choose to exploit the Windows CE native browser, and to extend it with plugins for visual rendering. Many of the most innovative and powerful technologies on the market has been reviewed, considering features like: (i) Availability, flexibility, and portability of the solution for a wide set of operating systems and HMI terminals; (ii) Vector graphics support for easy adaptation to different screen resolutions; (iii) availability of scripting languages for software personalization; and (iv) interactivity and usability features.

![Fig. 7. MyHMI Flash interface of the prototype.](image)

We considered technologies such as SVG [3], Adobe Flash, and Microsoft-based solutions. At the end of our evaluations, Adobe Flash [16] resulted to be the best candidate for the MyHMI client application.

The developed client-side prototypes include all the widgets needed to display plant values (e.g., gauge bars, alarm leds, conveyor belts, tachometers, meters, and so on) and to execute user commands (e.g., start/restart and emergency push buttons, input fields, and so on). Widgets have been designed according to the separation of concerns philosophy: every widget is designed separately, with an approach based on the well known concept of skin.

Thanks to the new widget design and to the sophisticated rendering engine provided by Flash, the prototype was able to implement much more refined interfaces than usual HMI systems, both in terms of graphical appearance and of business logics (for instance, see the real time conveyor emulator in Figure 7).

According to the decision presented in Section 4.1.5, the implementation strategy for the client run-time content adaptation and personalization exploited configuration rules compiled as binary code compiled at configuration time by the offline Configuration Tool. For compatibility with the Flash rendering engine, rules have been expressed as ActionScript code. Among the existing compilers of ActionScript, we choose MTASC [19], which provides binary code with better performances with respect to the...
competitors (see Section 7.2). MTASC is an open source project able to generate Flash SWF bytecode without relying on Adobe Flash components or other tools.

7. Evaluation

From a functional standpoint, our prototype covered most of the requirements presented in Section 3. The proposed architecture allows local, fixed-remote, and mobile-remote devices to effectively interact with server via standard HTTP connections. The adaptation subsystem allows a fine-grained specification of users and user groups, as well as content personalization and interface adaptation features.

The deployment device of choice for performance evaluation of our prototype implementation in real world scenario is an industrial panel equipped with a ViaX86 400MHz processor, 64MB of FLASH drive, 128MB of RAM, on Windows CE.Net 4.2.

7.1. Server-side evaluation

Tests performed over the server side of the MyHMI solution focused on verifying its ability to sustain multiple clients with different rendering technologies without an excessive degradation of the performances. The prototype application was able to score an average value of 6.3 requests/sec with 6 clients connected simultaneously. The client implementation technology was equally distributed between SVG and Flash (3 SVG clients and 3 Flash clients).

We also tested the performances when the client and the server reside on the same embedded device. The test revealed that MyHMI was able to perform similarly to the existing dedicated and monolithic ESA applications.

One of the most critical issues able to dramatically degrade the performance of the server is related to data format used for storing and managing persistent information. Modern Web technologies are heavily based on relational databases and XML documents for information gathering and exchange; however, a set of tests executed over the target hardware configuration showed that plain text file storage is still the best option for this class of devices.

As reported in Table 1 and Figure 8, we performed a detailed test using different data storage technologies according to the following conditions:

- For text files, read/write operations have been tested over a TXT file of 520 bytes; 1, 10 and 40 float values were processed respectively for the low, medium and high load.
- For XML files, low, medium and high load tests have been conducted with respectively 4, 14 and 20 Kbyte files containing 10, 40 and 100 complex objects (DOM and SAX parsers of Windows CE).
- For database technologies we tested Sql Server CE and SqLite by performing low, medium and high workload test with read/write operations respectively on 200 tuples with 10 columns each, 600 tuples, and 1000 tuples, both with strings and long integers values.

### Table 1. Performance evaluation comparison for data storage technologies (ms).

<table>
<thead>
<tr>
<th>Tech.</th>
<th>Oper.</th>
<th>Low load</th>
<th>Medium load</th>
<th>High load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RAM</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>XML</td>
<td>Read</td>
<td>5.3</td>
<td>5.5</td>
<td>16.1</td>
</tr>
<tr>
<td></td>
<td>SAX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read</td>
<td>59.2</td>
<td>469.8</td>
<td>80.4</td>
</tr>
<tr>
<td></td>
<td>DOM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>57.1</td>
<td>742.4</td>
<td>87.8</td>
</tr>
<tr>
<td></td>
<td>DOM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL CE</td>
<td>Read</td>
<td>148.8</td>
<td>143.5</td>
<td>377.1</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>46</td>
<td>235.7</td>
<td>46.4</td>
</tr>
<tr>
<td>SQ Lt</td>
<td>Read</td>
<td>16</td>
<td>19.1</td>
<td>28.4</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>13.5</td>
<td>525.2</td>
<td>13.4</td>
</tr>
<tr>
<td>Text</td>
<td>Read</td>
<td>0-1</td>
<td>0-1</td>
<td>0-1</td>
</tr>
<tr>
<td></td>
<td>Write</td>
<td>0-1</td>
<td>23.1</td>
<td>0-1</td>
</tr>
</tbody>
</table>

Figure 8. Performance results, Table 1 data (ms).

Each test has been executed over RAM memory (volatile) and FLASH memory (persistent) under the same conditions. Measured I/O time includes the time spent to open the data file, to execute reading or writing of data, and to close the file. We chose such testing condition to replicate typical workload conditions in real-world projects, in order to obtain a real picture of the performance of typical HMI devices.

Comparing the results obtained by performing standard read/write operations over text files, XML
files, and Embedded RDBMS solution, we discovered that plain text files could improve the overall performance of the system (both for main memory and solid read/write operations) with respect to XML files and to RDBMS technologies.

Likewise, our experiments showed that architectures based on Web services and/or XML document exchanges could worsen the performance of the server, which was in charge of building and parsing the exchanged documents. Plain text message composed by simple concatenated \textit{label:value} entries allowed to reduce the server processing time.

\subsection*{7.2. Client-side evaluation}

The tests performed on the client side for the chosen Flash implementation aimed at verifying: (i) the number of request per second that the client was able to send to the server; and (ii) the time needed to load a user interface page.

The first test has been carried out by exchanging 20 plant data at every request cycle with a remote Web server connected with a MODBUS simulator and deployed on a standard PC (in order to avoid bottlenecks due to limited resources available on a server installed on embedded devices and to simulate a real distributed environment). Thus, we deployed on the embedded industrial panel the only the Flash application.

Results obtained using standard Flash components show the Macromedia Flash application to be able of performing 3.2 requests/second using synchronous calls. Then, we manually optimized the application code, that resulted in performances almost doubled (we were able to reach up to 7.8 requests/sec). We obtained these results by removing all the Macromedia components dedicated to Web service connections which, whilst providing a convenient interface, bring heavy performance decay.

The second test we performed focuses on the page change elapsed time. The most demanding issues were related to the execution of the personalization and adaptation rules. For coming to the decision described in Section 4.1.5., we tested different solutions, including both compile-time and run-time user interface adaptation.

- The first approach uses f XML documents, dynamically generated by the server according to user and device rules to configure the client application. Such approach requires the client to dynamically build its interface by using, at run-time, the information parsed from the XML document.

- The second approach, instead, overturned such perspective by applying directly at client side the adaptivity and personalization rules by means of instructions hard-coded inside the client application and produced at compile-time when generating the Flash application.

Results showed that personalization rules applied client-side (written using ActionScript language and executed by the Flash files produced at compile-time) perform better than if applied by a server application written using C++ for embedded devices. In order to further improving the overall performances, we performed page-load time tests using, as compiling technologies, two frameworks freely downloadable from the Internet as open source projects: MTASC [19] and Laszlo [18].

Moreover, every tested Flash application has been executed on two identical devices, the first based on the Windows CE.Net 4.2 operating system, while the second working with Windows XP Embedded (service pack 2). Since the Flash player for Windows CE devices is not freely available from Adobe, a personalized version of the required Active-X (version: 7.0,70,3) was provided by NEC Corporation of America (NECAM) [17], running as a plug-in of Internet Explorer 6.0 for Windows CE. On Windows XP Embedded, we used the Adobe Flash player Active-X (version: 8,0,22,0) working as Internet Explorer 6.0 plug-in.

| Table 2. Performance evaluation comparison of different client technologies (ms) |
|-------------------------------|------------------|------------------|------------------|------------------|
| Tech.                        | Low load WinCE  | Low load XPe    | Medium load WinCE| Medium load XPe  | High load WinCE | High load XPe  |
| XML                          | 773             | 618             | 1472            | 1286             | 2910            | 2314            |
| Laszlo                       | 151             | 79              | 290             | 153              | 603             | 284             |
| MTASC                        | 87              | 58              | 153             | 107              | 276             | 192             |

The low load processed page was populated by 28 widgets, including 13 static objects, 11 simple read/write widget (e.g., push button) and 4 complex read/write widget (e.g., trend graph). Medium load and high load pages have been populated respectively with duplicating and quadruplicating the number of object considered for the first test.

Table 2 and Figure 9 report our results. The XML document approach resulted to be unacceptably slow both on WinCE and WinXP Embedded. Instead, we found that using the Laszlo technology the client configuration of each page run 5 times faster with respect to the XML approach, and the compiled application files, compared to the ones using XML, result to be 3 times smaller. The MTASC solution resulted even better then Laszlo and, hence, we chose it as the target client deployment platform.
We also compared the results of our architecture to the current state of the art in the market. MyHMI results to perform as fast as the average of the other HMI solutions based on the same hardware, although our solution provides much more powerful features. Only HMI devices with proprietary built-in O.S. could perform better.

Several research activities have gathered around the concept of Open-Architecture Controllers [13], focusing on software solutions that offer as much portability and openness to any kind of device and operating systems as possible. They aim at studying the best mix of programming languages, and development architectures and frameworks for granting such flexibility. They often foster Web technologies for their open availability and portability, but they don’t apply any effort for increasing the provided features or services.

Finally, another category of works, such as [14], provide experience reports on practical application of internet-based solutions to specific environments, but do not generalize their results to a general architecture for handling any kind of HMI problems.

Our contribution with respect to the current research is in the definition of a light-weight Web based software architecture that features standard technologies like browsers, Web servers, and graphic players, that can be delivered on (networks of) embedded systems, delivering state of the art Web applications, including personalization, multi-device adaptivity, and remote notification, without the need of heavy hardware platforms like office PCs.

9. Conclusion

In this paper we have shown how HMI systems could benefit from the introduction of cutting-edge Web technologies and best practices. The work has been organized in four sequential steps: market and literature analysis; definition of the requirements needed by new Web enabled HMI solutions; design of a proof-of-concept case study application; and development of the client and server applications to test reliability and performances of the MyHMI solution. The market analysis highlighted the current unsolved problems affecting the HMI market, thus gathering a set of features and requirements for the next generation of HMI solutions.

The result of our work consists in a highly configurable architecture which can be considered a state of the art reference for new generation of HMI solutions relying on flexibility, ubiquity and customization. Our contribution with respect to the current research is in the definition of a light-weight, Web based architecture that features standard technologies like browsers, Web servers, and graphic players, that can be deployed on (networks of) embedded systems, delivering state of the art Web applications. This architecture is able to provide

8. Related Work

The requirements, trends, and opportunities of current technological evolution in embedded system are widely recognized [7]. Some early works exist that describe how technologies from the Web environment can be applied to the industrial HMI applications. [8] outlines some possible approaches on how to use XML and Java for interface definition and configuration. However, our experience highlights that applying this plain technique can be painful in terms of performance if using low power devices with embedded operating systems.

Other works (e.g., [9], [11]) explore the integration of traditional field bus solutions with Ethernet based communication between clients, but, instead of proposing a full-fledged Web based architecture, they offer a gateway-based interface for transferring the information from the field to an office-like network. These approaches do not fully exploit the potentials that Web interfaces can provide (for example, in terms of richness of interfaces, adaptability and personalization), because the server-side software architecture is not structured enough for that. Moreover, they usually refer to office PC platform for running the advanced remote interfaces. Service-oriented, agent-oriented, and distributed object architectures ([10], [12]) based on Web and XML technologies have been explored too, but their results are still in an early phase of development and typically require powerful hardware.
advanced features including personalization, multi-
device adaptivity, and remote notification, without the
need of heavy hardware platforms like office PCs.

Future works will include some architecture
refinements and the testing of performances of
advanced features (messaging, remote logging, and so
on).

10. Acknowledgment

We wish to thank the staff of ESA Elettronica
S.p.A. for the valuable comments, feedbacks, and
collaborative work. A special thank to Stefano
Longoni (R&D department), to his team, and to Mario
Colombo (CEO).

11. References

[2] Extensible Markup Language (XML),
http://www.w3.org/XML/
[3] Scalable Vector Graphics (SVG),
http://www.w3.org/Graphics/SVG/
Web Tools to Monitor and Manage Embedded Devices”,
http://msdn.microsoft.com/msdnmag/issues/0500/wince/default
.aspx
view controller user interface paradigm in Smalltalk-80”, in
Journal of Object-Oriented Programming, Volume 1 , Issue 3
industrial automation--needs and opportunities,” in Automation
Architecture for HMI,” in: IEEE Conference on Systems, Man,
for industrial applications,” in: IEEE Transactions on
Instrumentation and Measurement, Feb 2003, Volume 52,
Issue 1, pp. 165- 174.
for industrial automation,” in IEEE Enterprise Distributed
Object Computing Conference, 2003, pp. 315 - 320
automation,” in: IEEE Computing & Control Engineering
[12] I. Seilonen, T. Pirttioja, P. Appelqvist, A. Halme, K. Koskinen,
“An Approach to Process Automation Based on Cooperating
Subprocess Agents,” chapter in: Holonic and Multi-Agent
Systems for Manufacturing, Springer LNCS, Volume 2744,
2004.
Manufacturing Systems,” Open Architecture Control Systems –
J.-E. Karlsson, “Human-computer interaction in flatness
control. An inraetnet-based approach,” in: IEEE Conference on
Systems, Man, and Cybernetics, 1998. Volume 2, pp. 1323-
1328.
automation,” in: IEEE Computing & Control Engineering
[17] NECAM (NEC Corporation of America),
http://www.necam.com
[19] MTASC (Motion-Twin ActionScript 2 Compiler)
http://www.osflash.org/mtasc
[20] Siemens Sm@rtAccess technology
ware/wincc-flexible-options/wincc-flex-smartaccess.htm
[22] W3C.org: HTTP1.1 Persistent connection (RFC 2616 Fielding,
et al.) http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html
[23] W3C.org “The XMLHttpRequest Object” (W3C Working
Draft) http://www.w3.org/TR/XMLHttpRequest/
http://www.laszlosystems.com/lps/docs/lzx-developers-
guide/persistent_connection.html
[25] Zhe Wang, Pei Cao, Persistent Connection Behavior of Popular
Browsers”, http://www.cs.wisc.edu/~cao/papers/persistent-
connection.html